DATE

Searching

Searching is a process of finding given values position in a list of values

Types of searching

1) Linear Search
2) Indexed Linear Search
3) Binary Search

1) Linear Search / Sequencial Search

It is a basic and simple search algorithm. It compares the element with all the other element given in the list. If the element is matched, it returns the index value, -1 otherwise.

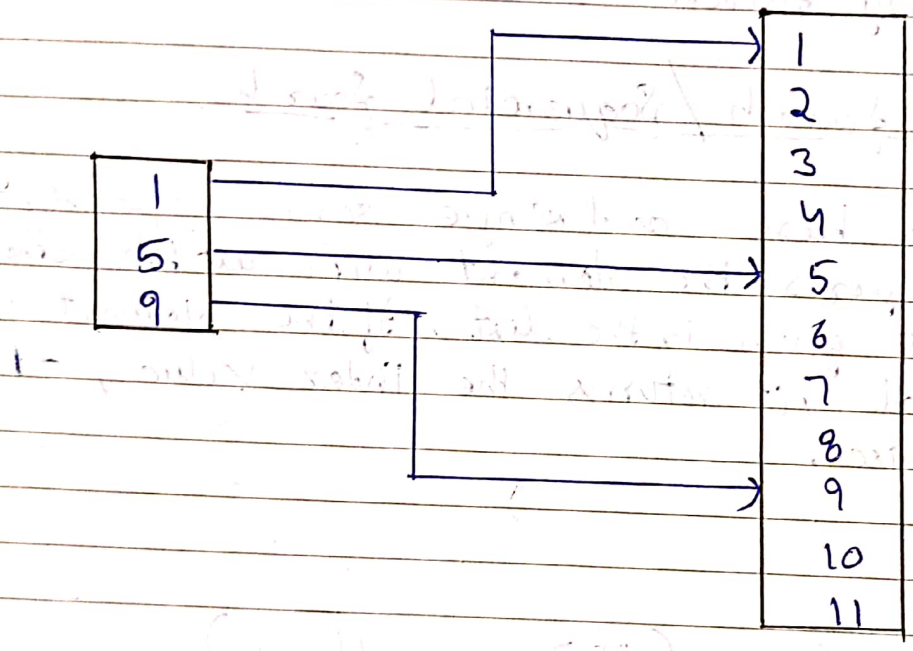
Algo:

```
LinearSearch (A[], key, lb, ub)
{
    for (k = lb , k <= ub ; k++)
    {
        if (key == A[k])
        {
            return k;
            break;
        }
    }
    return lb-1;
}
```

DATE

ii) Indexed linear Search

In this searching method, first of all an index file is created, that contains some specific group or division is required to record when the index is obtained then the partial indexing takes less time cause it is located in a specified group



DATE

iii) Binary Search

It is a searching algorithm used in a stored array by repeatedly dividing the search interval in half.
The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(log n).

```
BinarySearch ( A[ ], lb, ub, key )
{
    low = lb
    high = ub
    mid = (low + high) / 2
    while (A[mid] != key && low <= high)
    {
        if (item < A[mid])
            high = mid - 1;
        else
            low = mid + 1;
        mid = (low + high) / 2;
    }
    if (A[mid] == key)
        return mid;
    else
        return low - 1;
}
```

DATE

Hashing

Hash table

→ Hash table is one of the most important data structure that use a special function known as a hash function that map a given values with a key to access a element faster.

→ A hash table is a DS that stores some information, and the information has basically two main component i.e., key and values. The has table can be implemented with the help of an array.

→ The efficiency of mapping depends upon the hash function used for mapping

→ Complexity is always one.

function % no of element in array

Suppose the key value is x and the value is a phone number, so when we pass the key in the hash function the index is,

index = hash(x);

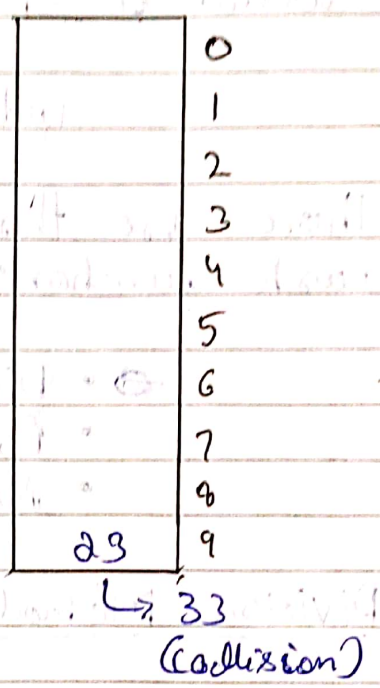
Drawback of hash function

A hash function assign each value with a unique key. Sometimes hash table use an imperfect function that passes a collision because the hash function generates the same key of two different value:

hash function = 2k+3

23 = (2(23)+3) % 10
= 49 % 10
= 9

33 = (2(33)+3) % 10
= 69 % 10
= 9



DATE

Hashing

→ It is one of the searching technique that use a ~~Constant~~ constant time. The time complexity in hashing is O(1)
~~The main function~~
→ The main idea behind the hashing is to create the key value pairs.
→ If the key is given then the algorithm computes the index at which the value would be stored. It can be written as:

index = hash (key)

There are three ways of calculating hash function:

- Division Method
- Folding Method
- Mod Square Method

Division Method

In the division method, the hash function can be defined as

$$h(K_i) = K_i \% m$$

where, i = no of keys
K = key
m = size of hash table
h = hash function

DATE

Collision

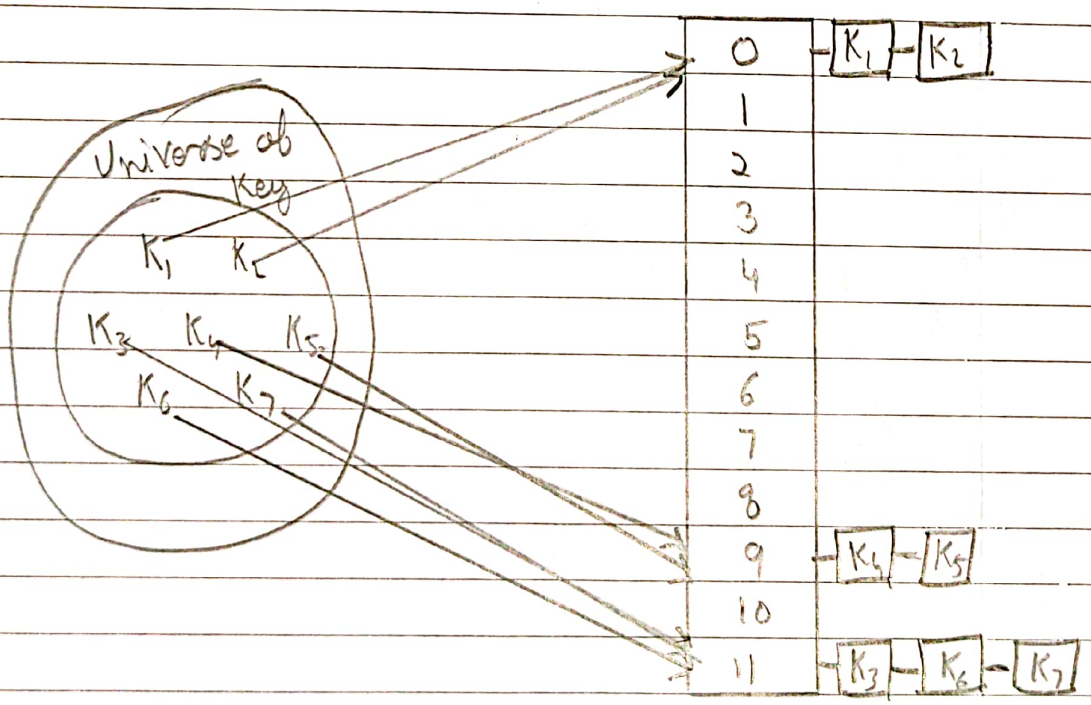
A collision occur when ~~more than~~ two peices of data in a hash teble share ~~the~~ the same hash value

The following are the collision techniques

→ open hashing (close addressing)
→ close hashing (open addressing)

Open hashing

In open hashing one of the method is used to resolve the collision is known as chaining method

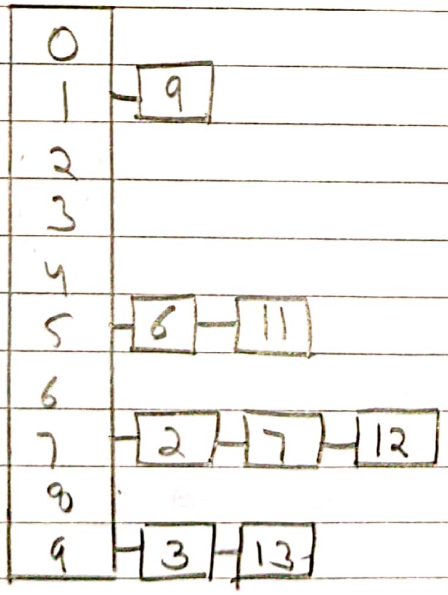


Example: Suppose we have a list of key values.

A = 3, 2, 9, 6, 11, 13, 7, 12

h(k) = 2k + 3 m = 10

key	location (index)
3	(2(3)+3)%10 = 9
2	(2(2)+3)%10 = 7
9	(2(9)+3)%10 = 1
6	(2(6)+3)%10 = 5
11	(2(11)+3)%10 = 5
13	(2(13)+3)%10 = 9
7	(2(7)+3)%10 = 7
12	(2(12)+3)%10 = 7



Closed hashing

In closed hashing three techniques are used to resolve a problem.

→ Linear probing

It is one of the forms of open addressing. As we know that each cell in the hash table contains a key value pair, so when the collision occurs by mapping a new key, then linear probing technique search for the closet free location and adds a new key to the empty cell. In this case, searching is preferred sequentially, starting from the position where the collision occur till the empty cell is not found.

Example: $A = \{3, 2, 9, 6, 11, 13, 7, 12\}$

$m = 10$, $h(k) = 2k + 3$

Prob	
1	$(2(3)+3) \% 10 = 9$
1	$(2(2)+3) \% 10 = 7$
1	$(2(9)+3) \% 10 = 1$
1	$(2(6)+3) \% 10 = 5$
2	$(2(11)+3) \% 10 = 5$ collision # 0
2	$(2(13)+3) \% 10 = 9$ collision # 0
2	$(2(7)+3) \% 10 = 7$ collision # 8
6	$(2(7)+3) \% 10 = 7$ collision # 2

13	0
9	1
12	2
	3
	4
6	5
11	6
2	7
7	8
3	9

→ Quadratic Probing

It is an open addressing technique that use quadratic polynomial for searching ~~until a empty~~ until a empty slot is found. It can be also defined as that it allows the insertion ki at first space from $(h+i^2)\%m$ where $i=0$ to $m-1$.

Example:

$A = \{3, 2, 9, 6, 11, 13, 7, 12\}$

$m = 10$; $h(k) = 2k+3$

$(3*2+3)\%10 = 9$

$(2*2+3)\%10 = 7$

$(2*9+3)\%10 = 1$

$(2*6+3)\%10 = 5$

$(2*11+3)\%10 = 5$ collision

when $i=0$

$(h+i^2)\%m = (5+0)\%10 = 5$

when $i=1$, $(5+1)\%10 = 6$

$((13*2)+3)\%10 = 9$ collision

when $i=0$, $(9+0)\%10 = 9$

when $i=1$, $(9+1)\%10 = 0$

$((7*2)+3)\%10 = 7$ collision

when $i=0$, $(7+0)\%10 = 7$

when $i=1$, $(7+1)\%10 = 8$

$(2*12+3)\%10 = 7$ collision

when $i=0$, $(7+0)\%10 = 7$

when $i=1$, $(7+1)\%10 = 8$

when $i=2$, $(7+4)\%10 = 1$

when $i=3$, $(7+9)\%10 = 6$

when $i=4$, $(7+16)\%10 = 3$

13	0
9	1
	2
12	3
	4
6	5
11	6
2	7
7	8
3	9

DATE

→ double probing

It is an open addressing technique which is used to avoid the collision ~~when~~ when the collision occurs then this technique use the secondary hash of the key. It use one hash value as an index to move forward until the empty location is found.

In double hashing two hash functions are used. Suppose $h_1(K)$ is one of the hash function is used to calculate the location. where as $h_2(K)$ is another hash function, it can be defined as to insert K_i at first free place from

$$(U + V * i) \% m,$$

where, $i = 1$ to $m-1$

U = location computed using the hash function $h_1(K)$

$V = h_2(K)$

Example:

$$A = \{3, 2, 9, 6, 11, 13, 7, 12\}$$

$$m = 10,$$

$$h_1(K) = 2k+3$$
$$h_2(K) = 3k+1$$

key	u	v	c
3	((2*3)+3)%10 = 9		1
2	(2*2+3)%10 = 7		1
9	(9*2+3)%10 = 1		1
6	(6*2+3)%10 = 5		1
11	(11*2+3)%10 = 5	(3*11+1)%10 = 4	3
13	(13*2+3)%10 = 9	(3*13+1)%10 = 0	
7	(7*2+3)%10 = 7	(3*7+1)%10 = 2	
12	(12*2+3)%10 = 7	(3*12+1)%10 = 7	2

	0
9	1
	2
11	3
12	4
6	5
	6
2	7
	8
3	9

for key = 11

i = 0

(5 + 4*0)%10 = 5

i=1 (5 + 4*1)%10 = 4

i=2 (5 + 4*2)%10 = 3

for key = 13

since v = 0, 9 will be returned in every iteration from i = 0 to m-1

∴, 13 can't inserted in hash table

for key = 7

i = 0, (7 + 2*0)%10 = 7

i = 1, (7 + 2*1)%10 = 9

i = 2, (7 + 2*2)%10 = 1

i = 3, (7 + 2*3)%10 = 3

i = 4, (7 + 2*4)%10 = 5

i = 5, (7 + 2*5)%10 = 7

i = 6, (7 + 2*6)%10 = 9

i = 7, (7 + 2*7)%10 = 1

i = 8, (7 + 2*8)%10 = 3

i = 9, (7 + 2*9)%10 = 5

since, we checked all places for 7 but no one is empty hence 7 cant be inserted into hash table

when key = 12

i = 0, (7 + 7*0)%10 = 7

i = 1, (7 + 7*1)%10 = 4

Sorting

Insertion Sort

```
for (i = 1; i < n; i++)
{
    temp = a[i];
    j = i-1;
    while (j >= 0 && a[j] > temp)
    {
        a[j+1] = a[j];
        j--;
    }
    a[j+1] = temp;
}
```

Complexity :-
Best : O(n)
Average : $O(n^2)$
Worst : $O(n^2)$

Implementation

Let
a[6] = {42, 29, 74, 11, 65, 58}

PASS 1: Assume that first element is sorted

42	29	74	11	65	58	29 < 42, insert 29 at index 0
42	42	74	11	65	58	
29	42	74	11	65	58	

PASS 2 :

29	42	74	11	65	58	42 < 74, No shifting

DATE

PASS 3

29 42 74 11 65 58 — 11 < 29, 11 < 42, 11 < 74, insert 11 at index 0
29 42 74 74 65 58
29 42 42 74 65 58
29 29 42 74 65 58
11 29 42 74 65 58

PASS 4

11 29 42 74 65 58 — 65 < 74 but 65 > 42, insert 65 at index 3
11 29 42 74 74 58
11 29 42 65 74 58

PASS 5

~~11 29 42 74 65~~
11 29 42 65 74 58 — 58 < 74, 58 < 65 but, 42 < 58 insert 58 at index 3
11 29 42 65 74 74
11 29 42 65 65 74
11 29 42 58 65 74

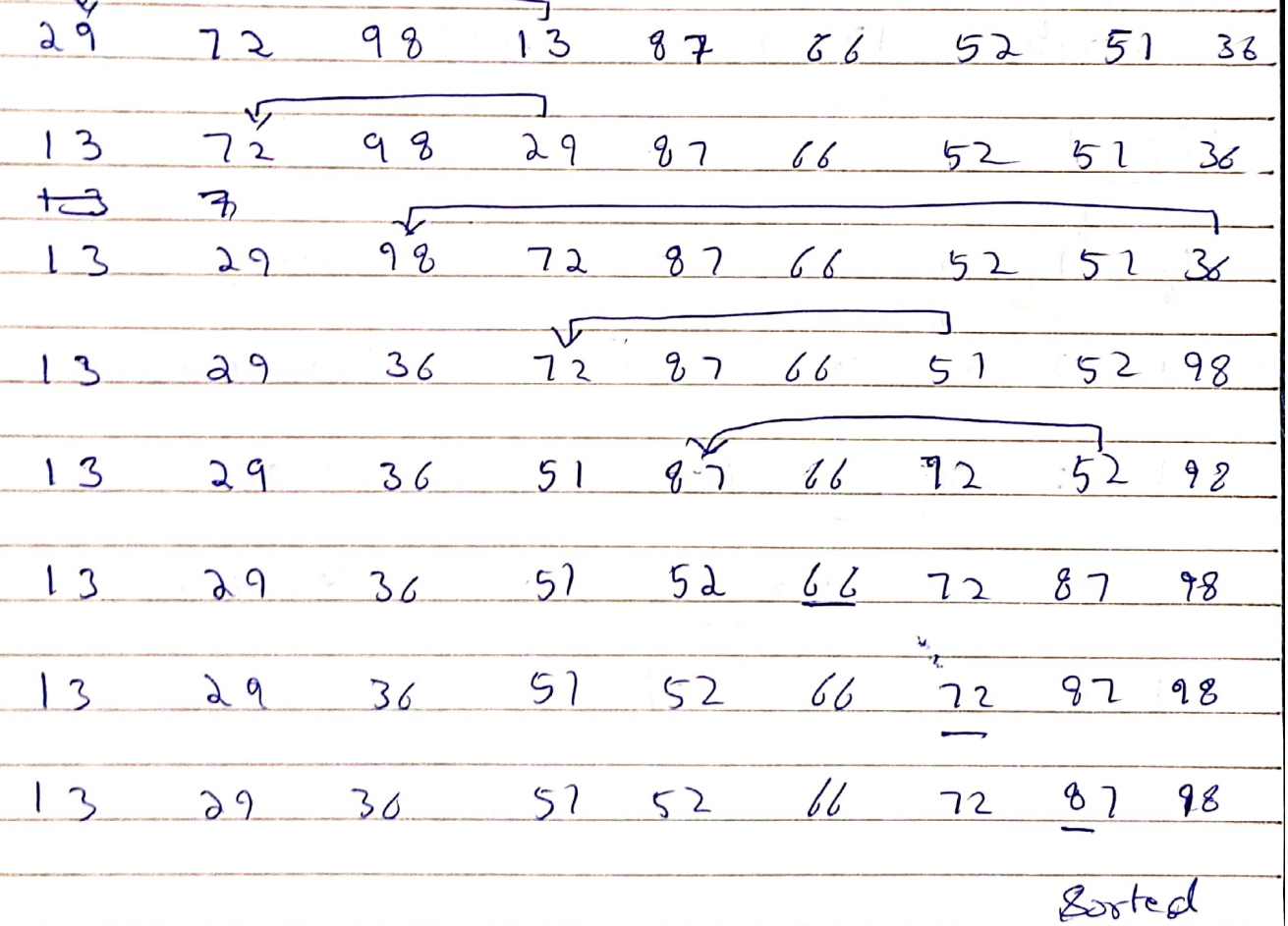
Sorted Array is : ~~11 29~~ 11 29 42 58 65 74

DATE

Selection Sort

```
for (i=0 ; i<n; i++)
{
    small = i;
    for (j=i+1 ; j<n ; j++)
        if (arr[j] < arr[min])
            small = j
    temp = arr[i]
    arr[i] = arr[min]
    arr[min] = temp
}
```

Implementation

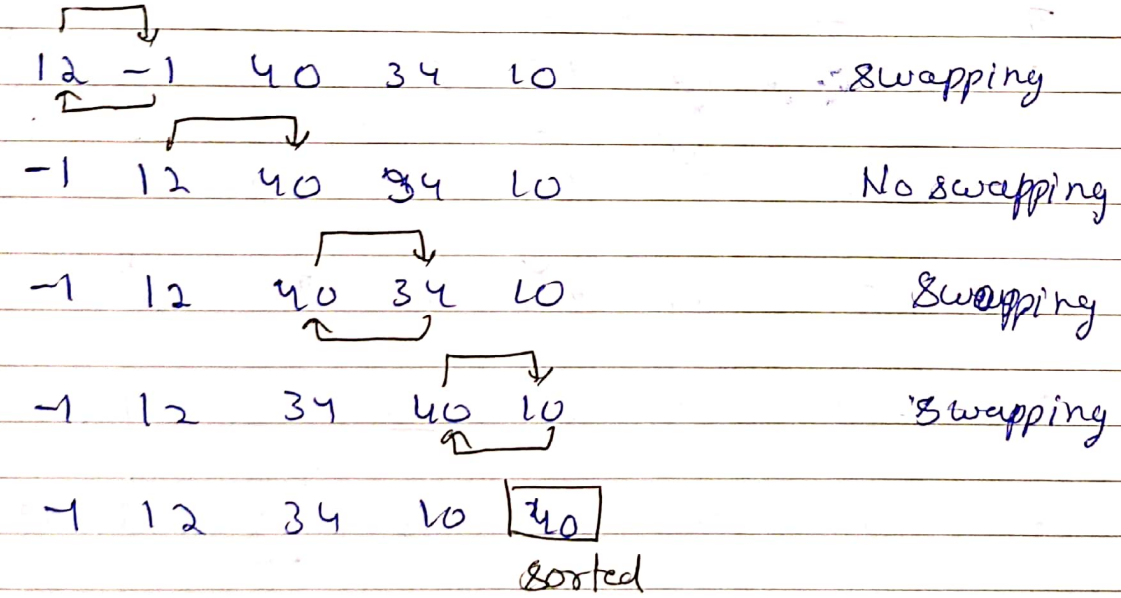


Bubble Sort

```
for (i=0; i<n; i++)
{
    for(j=0; j<n-i-1; j++)
    {
        if (arr[j] > arr[j+1])
        {
            temp = arr[j]
            arr[j] = arr[j+1]
            arr[j+1] = temp
        }
    }
}
```

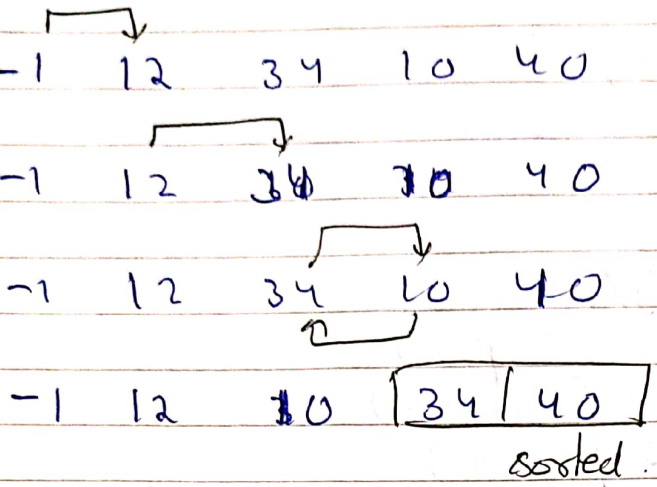
arr[5] = { 12, -1, 40, 34, 10}

PASS 1

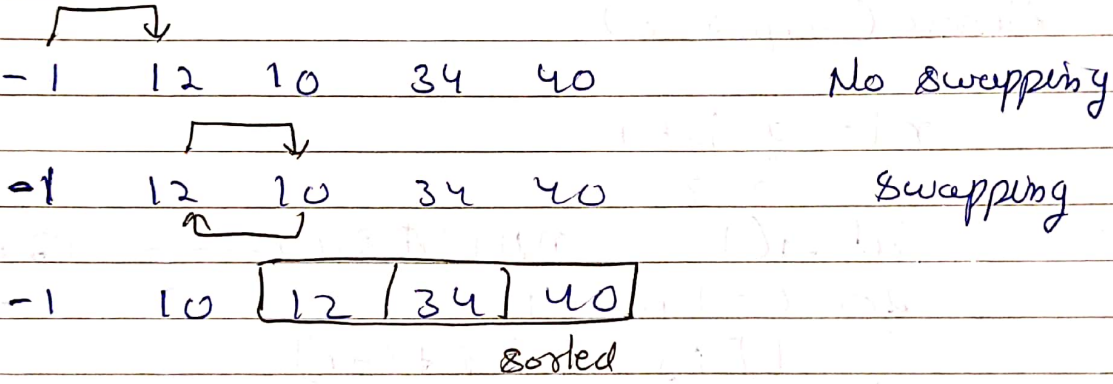


DATE

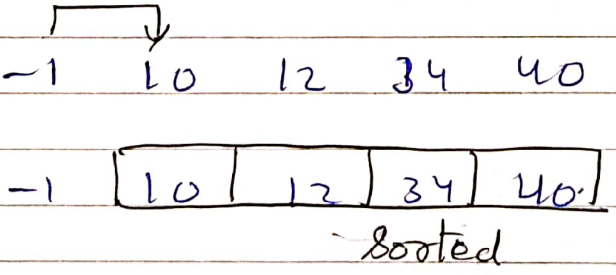
PASS 2



PASS 3



PASS 4



DATE

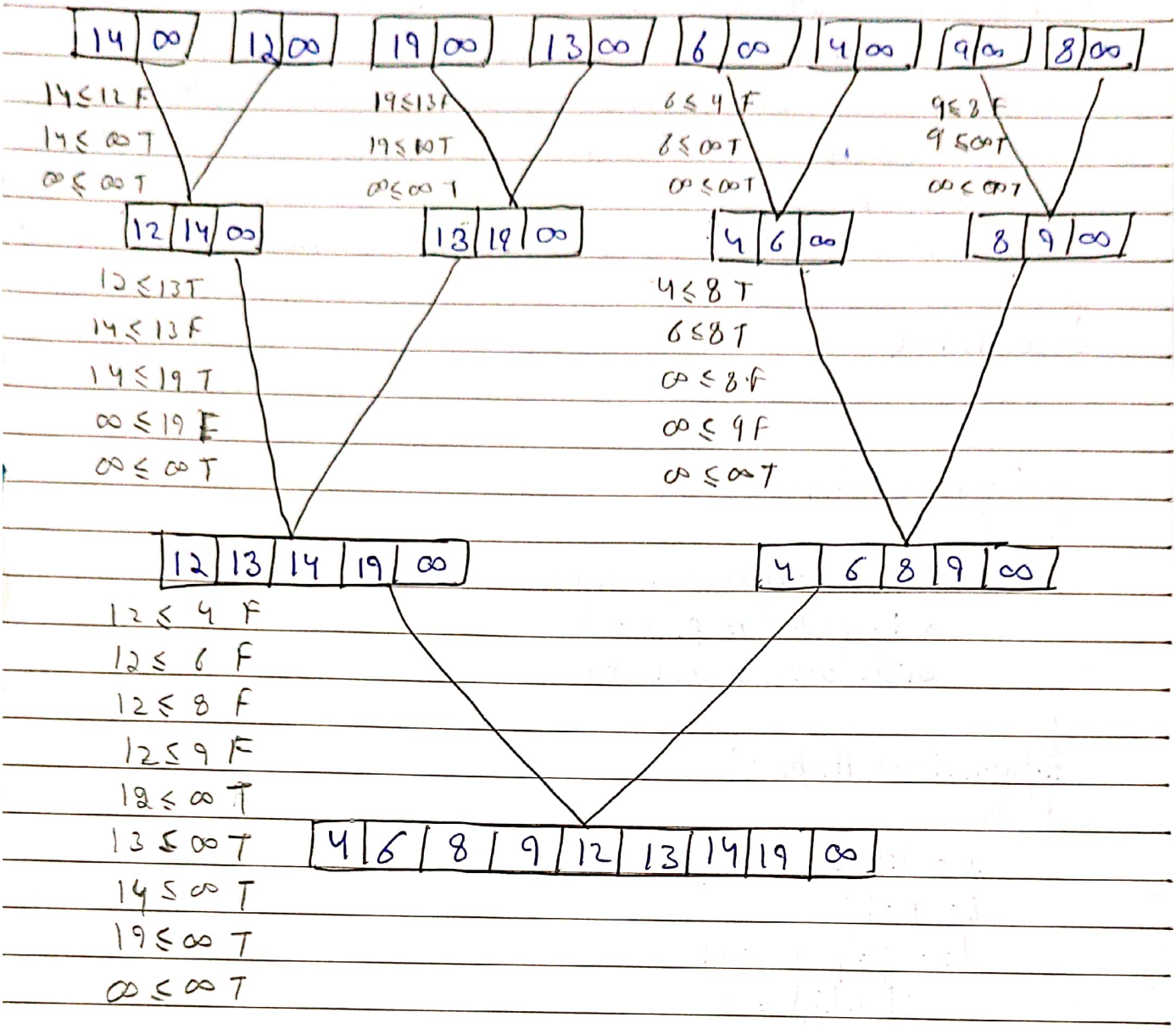
Merge Sort

```
Merge Sort (A, p, r)
{
    if (p < r)
    {
        q = (p+r)/2
        Merge Sort (A, p, q)
        Merge Sort (A, q+1, r)
        Merge (A, p, q, r)
    }
}
Merge (A, p, q, r)
{
    n1 = q - p + 1
    n2 = r - q
    let L[1 ... n1+1] & R[1 ... n2+1]
    for i = 1 to n1
        L[i] = A[p+i-1]
    for j = 1 to n2
        R[j] = A[q+j]
    L[n1+1] = ∞
    R[n2+1] = ∞
    i = 1 , j = 1
    for ~~L[i] <= R[j]~~
        ~~A[k] = L[i]~~
        ~~i = i+1~~
    for k = p to r
    { if L[i] <= R[j]
        {A[k] = L[i]
         i = i + 1}
      else
        {A[k] = R[j]
         j = j + 1}
    }
}
```

DATE

Example

14 12 19 13 6 4 9 8



DATE

Quick Sort

→ This algorithm follows divide and conquer approach, Divide and conquer is a technique of breaking down the algorithms into sub group, the solving the sub group and combining the results back together to solve the original problem

→ Quick Sort picks an pivot element and the it partitions the given array around the picked pivot element

Algorithm

```
Quick Sort (A, p, r)
{ if p < r
      q = partition (A, p, r)
      Quick Sort (A, p, q-1)
      Quick Sort (A, q+1, r)
}
partition (A, p, r)
{
    x = A[r]
    i = p-1
    for j = p to r-1
        if A[j] ≤ x
            i = i+1
            exchange A[i] with A[j]
    exchange A[i+1] with A[r]
    return i+1
}
```

DATE

A[r] = x

24, 9, 29, 14, 19, 27

j = 1 , A[r] = x = 27

A[j] <= x

24 <= 27 True

i = 0+1 = 1

exchange A[i] with A[j]

24, 9, 29, 14, 19, 27

j = 2 , A[r] = x = 27

A[j] <= x

9 <= 27 True

i = 1 + 1 = 2

exchange A[i] and A[j]

9, 24, 29, 14, 19, 27

j = 3 , A[r] = x = 27

A[j] <= x

29 <= 27 false

9, 24, 29, 14, 19, 27

j = 4, A[r] = x = 27

A[j] <= x

14 <= 27 True

i = 2+1 = 3

exchange A[i] with A[j]

DATE

```
          i       j       x
          ↓       ↓       ↓
9, 14, 29, 24, 19, 27
```

j = 5, A[r], x = 27

A[j] <= x

19 <= 27 True

i = 3+1 = 4

Exchange A[i], A[j]

```
          i               x
          ↓               ↓
9, 14, 19, 24, 29, 27
```

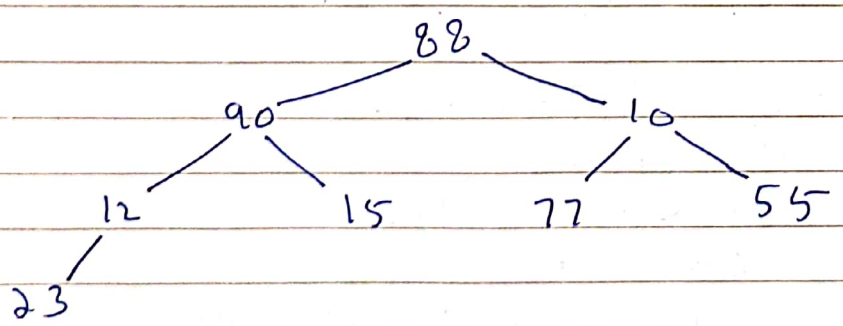
Exchange A[i+1] with A[r]

9, 14, 19, 24, 27, 29

Heap Sort

A heap is complete binary tree, A binary tree is a tree in which the node can have the utmost two children. A complete binary tree is a binary tree in which every level is completely filled and all nodes in the last level are as far left as possible

eg; 88, 90, 10, 12, 15, 77, 55, 23



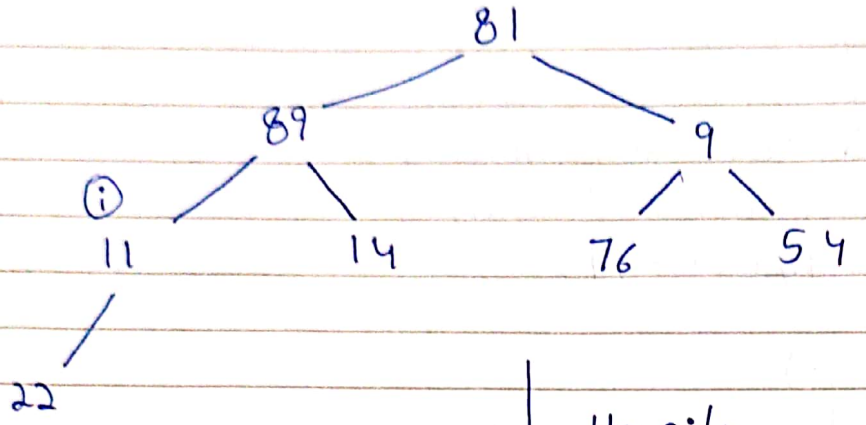
DATE

Algorithm

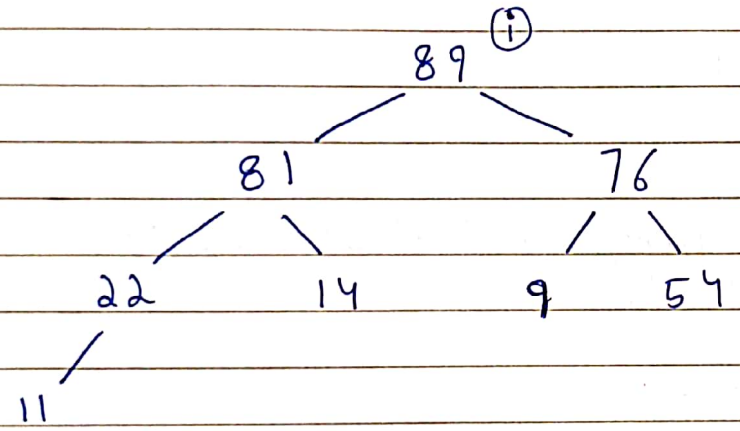
```
Heap Sort ( Arr)
    Build - Max - heap (Arr)
    for i = length(Arr) to 2
        Swap Arr [1] with Arr[i]
        heap size [arr] = heap size [arr] - 1
        Max heapify (Arr, 1)

Built Max Heap ( Arr )
    heap size [Arr] = length [Arr]
    for i = (length [arr] /2) to 1
        Max heapify (Arr, i)

Max heapify (Arr, i)
    L = left [i]
    R = right [i]
    if (L < heapsize [Arr] and Arr [L] > arr [i])
        largest = L
    else
        largest = i
    if ( R < heapsize [Arr] and Arr [R] > Arr [largest])
        largest = R
    if largest != i
        Swap Arr[i] with Arr[ largest]
    Max heapify [Arr, largest]

END
```

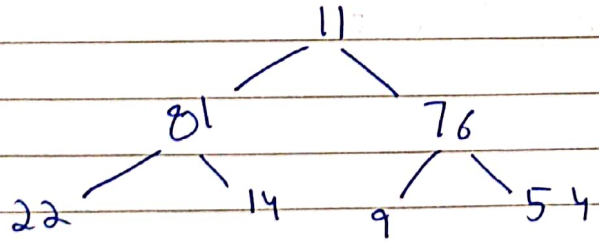
DATE

Example : 81, 89, 9, 11, 14, 76, 54, 22

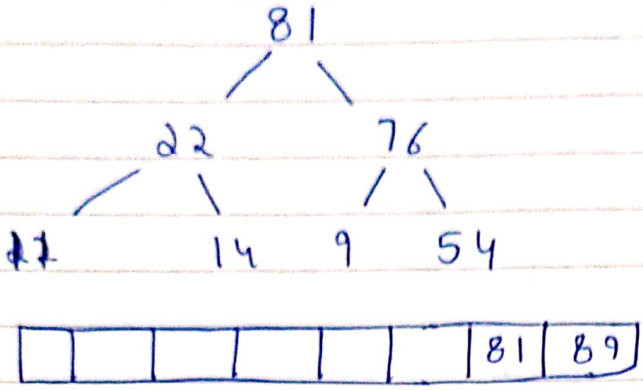


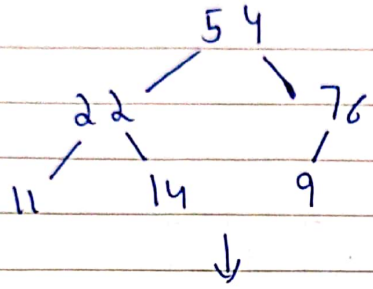
Heapify $i = 8/2 = 4$

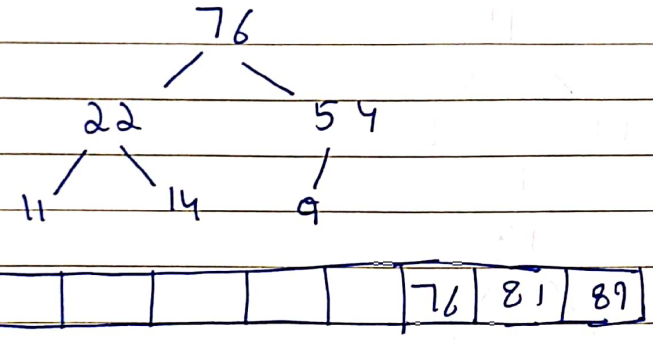


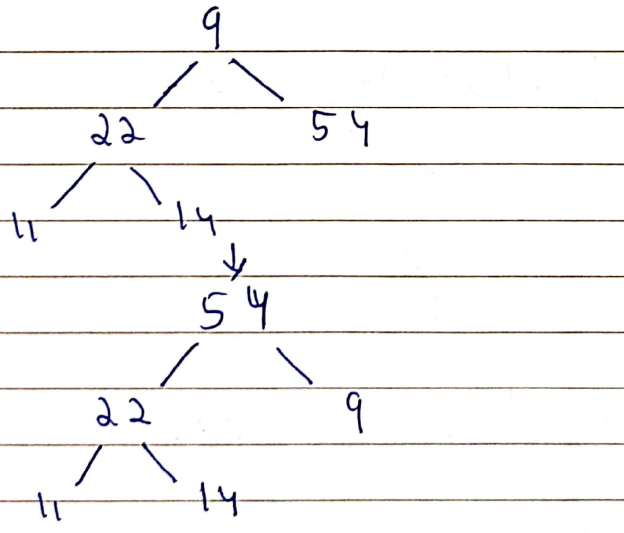


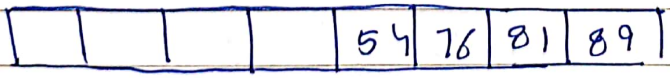
DATE


81
22
76
11
14
9
54
81
89

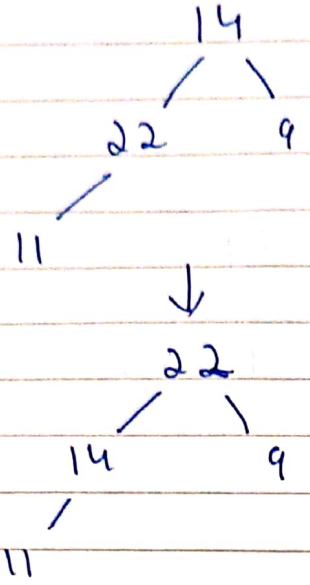

54
22
76
11
14
9

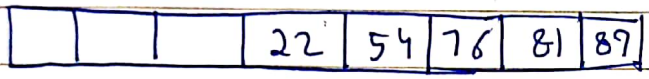

76
22
54
11
14
9
76
81
89

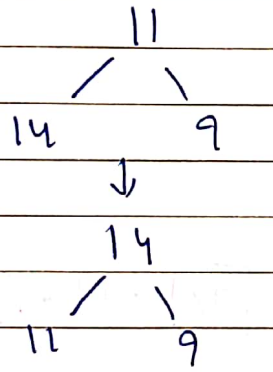

9
22
54
11
14
54
22
9
11
14

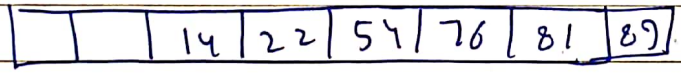

54
76
81
89

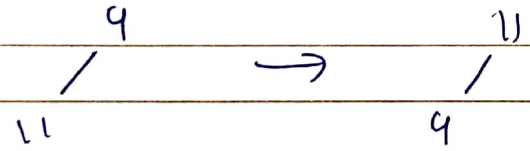
DATE

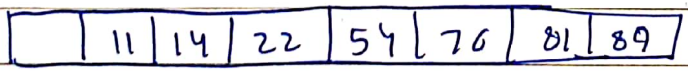

14
22
9
11
22
14
9
11


22
54
76
81
89

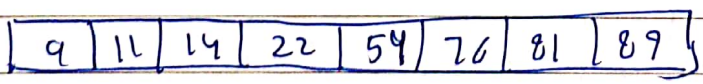

11
14
9
14
11
9


14
22
54
76
81
89


9
11
11
9


11
14
22
54
76
81
89

9


9
11
14
22
54
76
81
89